# 📄 OpenAPI Specification Documentation Guide

This guide outlines the structure and conventions for writing and maintaining OpenAPI specifications for the TestStack project. It includes directory organization, naming conventions, and a list of implemented API specs.

## 🗂️ Directory Structure

```
openapi/
├── openapi.yaml
├── paths/
│   ├── v1/
│   │   ├── projects/
│   │   ├── imports/
│   └── v2/
│       └── projects/
├── components/
│   ├── parameters/
│   │   ├── ProjectIdParam.yaml
│   │   ├── FolderIdParam.yaml
│   │   └── TestCaseIdParam.yaml
│   ├── schemas/
│   │   ├── Project.yaml
│   │   ├── TestCase.yaml
│   │   ├── PageInfo.yaml
│   │   └── Comment.yaml
│   └── responses/
│       ├── BadRequest.yaml
│       ├── Unauthorized.yaml
│       ├── Forbidden.yaml
│       ├── NotFound.yaml
│       ├── UnprocessableEntity.yaml
│       └── InternalServerError.yaml
```

## 🗒️ File Naming and Structure

### 1. Path Files ( `paths/v1/projects/...` , `paths/v2/projects/...` )

- File names are based on the full endpoint path.
- Replace all slashes ( `/` ) with underscores ( `_` ).
- Enclose all path parameters in curly braces ( `{}` ), e.g., `{project_id}` , `{folder_id}` .
- Use **lowercase** and **underscores** only.
- **Do not** include the HTTP method ( `_get` , `_post` , etc.) in the file name.

#### 📘 Example Endpoint

**URL**

`GET /v1/projects/{project_id}/folder/{folder_id}/testcases/{test_case_id}/comments`

#### 📁 File Name

`api_v1_projects_{project_id}folder{folder_id}testcases{test_case_id}_comments.yaml`

### 2. Parameter Files ( `components/parameters/` )

- Each commonly used path or query parameter is defined **once** in this directory for reuse across endpoints.
- File names follow the format:

```
<ParameterName>Param.yaml
```

- Use **PascalCase** for the parameter name and append `Param`.
- Parameters include the `name`, `in` (`path`, `query`, etc.), `required`, `schema`, and `description`.

**Examples:**

- `ProjectIdParam.yaml` → defines the `{project_id}` path parameter.
- `FolderIdParam.yaml` → defines the `{folder_id}` path parameter.
- `TestCaseIdParam.yaml` → defines the `{test_case_id}` path parameter.

### 3. Schema Files ( `components/schemas/` )

- Define all **reusable data models** used in request and response bodies.
- Schemas should be **modular**, reusable, and placed in separate YAML files.
- Use **PascalCase** for file names (e.g., `Project.yaml`, `Folder.yaml`, `TestCase.yaml`).
- **Do not** include example data inside schema files.
- **Do not** use schemas directly in responses — they must be referenced via **response files** under `components/responses/`.

### 4. Response Files ( `components/responses/` )

- Contains **standard response wrappers** and **successful responses** (such as 200 OK).
- Each file represents a **common HTTP status code** response or a **successful response**.
- File names should be in **PascalCase** (e.g., `BadRequest.yaml`, `Unauthorized.yaml`, `SuccessResponse.yaml`).

## 🧩 Adding a New API to the OpenAPI Specification

Follow these steps to correctly add a new API to the OpenAPI YAML specification:

### 1. Reference the API in `openapi.yaml`

Add a reference entry for the new API path in the `paths` section of `openapi.yaml`.

**Example:**

```yaml
paths:
  /v1/projects/{project_id}/new-endpoint:
    $ref: paths/v1/projects/api_v1_projects_{project_id}_new-endpoint.yaml
```

### 2. Create the Path File

Create a corresponding YAML file inside the correct folder in the `paths/` directory.
Use lowercase names and replace `/` with `_` and include path parameters in `{}`.

**Example:**

**File Path:**
`paths/v1/projects/api_v1_projects_{project_id}_new-endpoint.yaml`

**File Content:**

```yaml
get:
  summary: Get information about the new endpoint
  parameters:
```

```
    - $ref: ../../../components/parameters/ProjectIdParam.yaml
  responses:
    '200':
      description: Successful response
      content:
        application/json:
          schema:
            $ref: ../../../components/responses/NewEndpointResponse.yaml
    '404':
      $ref: ../../../components/responses/NotFound.yaml
```

### 3. Create Schema and Response Files

#### 📦 Schema File

Create a reusable response schema in `components/schemas`.

**File Path:**

`components/schemas/NewEndpointResponse.yaml`

**Example:**

```
type: object
properties:
  id:
    type: integer
  name:
    type: string
  status:
    type: string
    enum: [active, inactive]
required:
  - id
  - name
  - status
```

#### 📬 Response File

Create or reuse a standard response file in `components/responses`.

**File Path:**

`components/responses/NewEndpointResponse.yaml`

**Example:**

```
description: Successful response for New Endpoint
content:
  application/json:
    schema:
      $ref: ../schemas/NewEndpointResponse.yaml
```

This ensures your schema is reusable and keeps response files clean and maintainable.

### 4. Bundle and Validate

Once all files are created, use Redocly CLI to bundle the OpenAPI specification and check for errors.

**Command:**

`redocly bundle /your/path/to/openapi/openapi.yaml -o openapi.yaml`

This will bundle your OpenAPI specification and check for any errors or issues.

This is a small example, but the same steps apply to larger APIs with more complexity. You just need to add the appropriate parameters, response schemas, and handle any additional endpoints in the same manner.

## ✅ Best Practices

- **Extract Shared Fields and Structures into Components**
  Always extract common parameters, schemas, and responses into the `components/` directory. This reduces duplication and improves maintainability.
- **Use `$ref` for Reusability**
  Use `$ref` to reference shared components (parameters, schemas, responses) across your spec. This ensures consistency and easier updates.
- **Logical Grouping by Resource**
  Organize path files by resource. For example, keep all project-related files inside `paths/v1/projects/`. This improves navigation and structure.
- **Use PascalCase for Component File Names**
  All files under `components/parameters`, `components/schemas`, and `components/responses` should follow PascalCase naming for consistency and clarity (e.g., `ProjectIdParams.yaml`, `TestCaseSchema.yaml`, `NotFound.yaml`).
- **Replace Slashes with Underscores in Path Filenames**
  Convert all `/` to `_` in path filenames and enclose parameters in `{}` (e.g., `/v1/projects/{project_id}/edit` → `api_v1_projects_{project_id}_edit.yaml`).
- **No Inline Schemas in Responses**
  Always define response schemas in `components/schemas/` and reference them in response files. Do not define schemas directly inside response files.

## 🔍 Covered API Endpoints

- `/v1/projects`
- `/v1/projects/{project_id}`
- `/v1/projects/{project_id}/edit`
- `/v1/projects/{project_id}/delete`
- `/v1/projects/{project_id}/configurations`
- `/v1/projects/{project_id}/users`
- `/v1/projects/{project_id}/dashboard-analytics/active-test-runs-info`
- `/v1/projects/{project_id}/dashboard-analytics/closed-test-runs-info`
- `/v1/projects/{project_id}/dashboard-analytics/closed-test-runs-split`
- `/v1/projects/{project_id}/dashboard-analytics/test-case-type-split`
- `/v1/projects/{project_id}/dashboard-analytics/test-case-count-trend`
- `/v1/projects/{project_id}/dashboard-analytics/automation-stats`
- `/v1/projects/{project_id}/dashboard-analytics/issues-count-info`
- `/v1/projects/{project_id}/test-cases`
- `/v1/projects/{project_id}/test-cases/archive_count`
- `/v1/projects/{project_id}/test-cases/bulk-copy`
- `/v1/projects/{project_id}/test-cases/{test_case_id}/detail`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/{test_case_id}`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/{test_case_id}/detail`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/{test_case_id}/edit`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/{test_case_id}/delete`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/bulk-edit`

- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/bulk-delete`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/bulk-move`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/bulk-archive`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/{test_case_id}/comments`
- `/v1/projects/{project_id}/folder/{folder_id}/test-cases/{test_case_id}/comments/{comment_id}`
- `/v1/import/xray/quick/insert`
- `/v1/import/xray/quick/test-connection`
- `/v1/import/xray/quick/{import_id}/retry`
- `/v1/imports`
- `/v1/import/latest`
- `/v1/integration/tcg/test-generation/cancel-suggest-test-cases`
- `/v1/integration/tcg/test-generation/fetch-test-case-details`
- `/v1/integration/tcg/test-generation/suggest-test-cases`
- `/v1/integration/tcg/test-management/suggest-description`
- `/v1/integration/tcg/test-management/suggest-test-steps`
- `/v1/integration/tcg/test-management/suggest-preconditions`
- `/v1/integration/tcg/genrate-token`
- `/v1/projects/{project_id}/reports/schedules`
- `/v1/projects/{project_id}/reports`
- `/v1/projects/{project_id}/reports/{report_id}`
- `/v1/projects/{project_id}/reports/{report_id}/detail/{report_type}`
- `/v1/projects/{project_id}/reports/schedules/{schedule_id}`
- `/v1/projects/{project_id}/reports/{report_id}/download`
- `/v1/projects/{project_id}/reports/attachments/{attachment_id}`
- `/v1/projects/{project_id}/reports/{report_id}/selection/edit`
- `/v1/projects/{project_id}/test-plans`
- `/v1/projects/{project_id}/test-plans/{test_plan_id}`
- `/v1/projects/{project_id}/test-plans/{test_plan_id}/test-runs`
- `/v1/projects/{project_id}/test-plans/{test_plan_id}/test-runs/unlink`
- `/v1/projects/{project_id}/test-plans/{test_plan_id}/update`
- `/v1/projects/{project_id}/test-plans/{test_plan_id}/delete`
- `/v1/projects/{project_id}/datasets`
- `/v1/projects/{project_id}/datasets/variables`
- `/v1/projects/{project_id}/datasets/{uuid}`

- `/v2/projects`
- `/v2/projects/{project-id}`